Dynamo: A Runtime Codesign Environment

Heather Quinn[1], Miriam Leeser
Dept of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
{hquinn, mel}@ece.neu.edu

L. A. Smith King
Dept. of Mathematics and Computer Science
College of the Holy Cross
Worcester, MA 01610
LA@cs.holycross.edu

Systems using Field Programmable Gate Array (FPGA) boards have been proven effective for gaining one to three orders of magnitudes speedup over systems based solely on PCs. Signal and image processing applications are especially attractive for implementation on FPGAs as their computationally intensive and massively parallel algorithms can effectively take advantage of the FPGA architecture. In moving part of an algorithm to hardware, one must consider overhead costs as well as the improvement in the computation to determine whether the move will result in overall system speedup. Dynamo is a runtime system for generating hardware/software pipeline implementations. Dynamo balances the benefits of hardware and software implementations and takes overhead costs into account in order to accurately predict runtimes of hardware/software systems.

Currently, the Dynamo system generates hardware/software solutions for image processing applications from a library of predefined component implementations. Many image processing applicationsconsist of a series of algorithms applied to the image, forming a computation *pipeline*. When using Dynamo, an image analyst only needs to specify the pipeline of image processing components to apply and an image or images to process. Image processing components are chosen from a library of predefined components: the *image processing Basic Library of Components* (ipBLOC). Each component in the library has at least one software and one hardware implementation associated with it. From the pipeline specification, Dynamo selects the most efficient combination of of hardware or software component implementations to minimize pipeline runtime (*pipeline assignment*), generates the source code for a HW/SW implementation of the pipeline (*pipeline compilation*), processes the input image(s) using the generated pipeline (*pipeline execution*), and returns the result to the analyst.

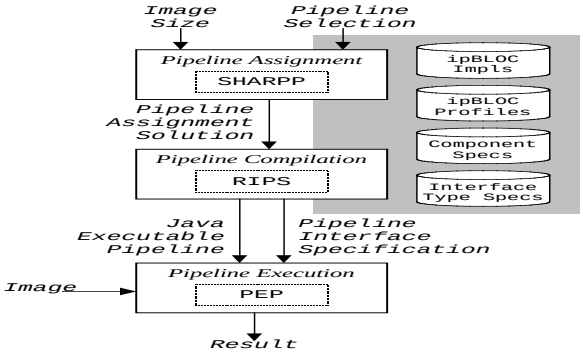


Figure 1: System Overview

The goals of the Dynamo system are to allow an image analyst to focus solely on pipeline selection, create image processing pipelines at runtime, make efficient use of software and FPGA hardware, and build pipelines that maximize performance. Currently, Dynamo is restricted to image processing applications, but is easily extensible to other application domains. Dynamo's design includes three major subsystems which implement pipeline assignment, pipeline compilation and pipeline execution, respectively. These subsystems

[1]Heather Quinn is supported under a Graduate Student Researchers Program Fellowship through NASA's Goddard Space Flight Center.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 FEB 2005	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Dynamo: A Runtime Codesign Environment	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Electrical and Computer Engineering Northeastern University Boston, MA 02115**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004. , The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**6**	

interact with Dynamo's predefined libraries, including the ipBLOC. Figure 1 shows an overview of the Dynamo system.

At the core of the Dynamo runtime system is *pipeline assignment* (PA). PA assigns hardware or software implementations to each component in the pipeline to minimize total pipeline execution time, while meeting the area requirements for the FPGA device. There are costs associated with passing intermediate images and data between components that differ depending on component types and interfaces. These costs include data movement, communication, reprogramming and hardware initialization. PA must balance the latency and area costs to find the best solution for a given FPGA device. Profiles of the component implementations and of overhead runtimes allow PA to accurately predict pipeline solution runtimes. The pipeline assignment problem is NP-complete, and is similar to the hardware/software codesign partitioning problem.

The Dynamo subsystem responsible for solving PA is named SHARPP (Software/HArdware Runtime Procedural Partitioning). SHARPP solves the pipeline assignment problem quickly at runtime and uses different algorithms for different pipeline sizes to keep execution time short. SHARPP uses dynamic programming for small pipelines (1-7 components) and two variations of tabu search to solve large pipelines (7-20 components). Tabu search is a metahueristic based on local search which provides near-optimal PA solutions. The two variations find solutions that are on average 18% slower than the optimal pipeline latency.

Pipeline compilation and execution occur within the runtime environment. Dynamo's RIPS (Runtime Interfacing for Pipeline Synthesis) subsystem performs *pipeline compilation*. RIPS uses the SHARPP output to compose an executable that calls the appropriate implementations and overhead methods that comprise a pipeline solution. The pipelines are constructed at runtime because pre-constructing PA solutions for all possible combinations of component implementations is not practical. Finally, the pipeline is executed. Dynamo's execution system runs the compiled pipeline on the selected image and outputs the results.

As an illustration of how Dynamo works, assume the image analyst has chosen to run the pipeline "median filter → histogram" on a 40185-pixel image. Table 2 shows the four possible solutions to this pipeline and the SHARPP predicted latencies. The component assignments are shown within parentheses with ":sw" indicating software and ":hw" indicating hardware. These results show that the optimal solution for this image size uses the hardware implementations for both components. Figure 2 shows what the pipeline looks like when the overhead costs are included in the pipeline, where the squares represent the components and the circles represent the overhead methods. RIPS was used to build executable solutions for each of the four pipelines. These pipelines were executed and measured so the actual execution times could be compared with SHARPP's estimates. The relative error for the predicted latency is quite low.

We have tested pipelines that range in size from one to 20 components. Many short pipelines are completely assigned to software since the hardware initialization cost is relatively high. As pipelines included more components, enough speedup will be gained by processing in hardware to mitigate the device initialization cost. Therefore, longer pipelines tend to be mainly assigned to hardware, except when either the image is small or the component is not well matched to the FPGA architecture.

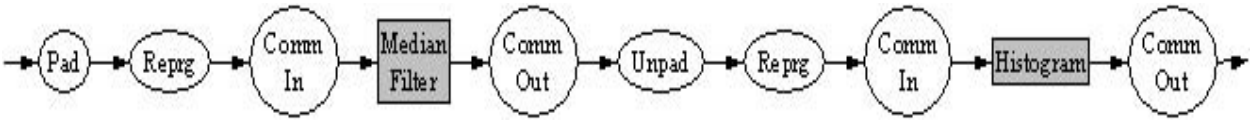


Figure 2: Pipeline Solution with Overhead Methods

Annotated Solution	Predicted Runtime (ms)	Actual Runtime (ms)	Relative Error
(mf:sw) → (histogram:sw)	3766	3801	0.009
(mf:sw) → (histogram:hw)	6076	5844	-0.038
(mf:hw) → (histogram:sw)	2772	2836	0.023
(mf:hw) → (histogram:hw)	2718	2260	-0.169

Table 2: The Four Solutions to median filter → histogram

Dynamo: A Runtime Codesign Environment

Heather Quinn[1], Dr. Miriam Leeser
Northeastern University
[1]hquinn@ece.neu.edu

Dr. Laurie Smith King
College of the Holy Cross

Abstract

Agenda

Motivation: Accelerate image processing tasks through efficient use of FPGAs. Combine already designed components at runtime to implement series of transformations (pipelines)

Fast, Flexible Image Processing

Run this pipeline:

On this Environment:

- Which component implementations to use?
- How to minimize overall latency?
- When to use FPGA?
- How to change the pipeline or interfaces dynamically?

Reconfigurable Systems

- Using reconfigurable hardware incurs execution costs not present in software or ASIC-based systems
  - Hardware initialization
  - Communication
  - Reprogramming

Efficient Use of FPGAs

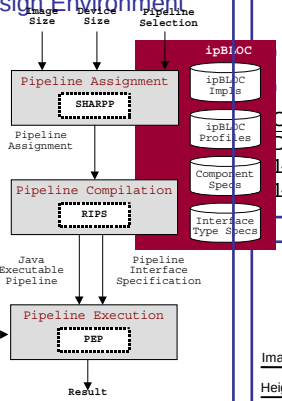
- Software algorithm's runtime for small images less than the hardware costs
  - Profiling the hardware and software runtimes for different image sizes determines the crossover point
  - Deciding at runtime to execute in software or hardware is simple for one algorithm processing one image

Image Processing Pipelines

- Series of image processing algorithms applied to an image
- Each algorithm has a software and hardware implementation
- Finding the optimal pipeline assignment is complicated
  - Exponential number of implementations
  - Coupling costs differ for each pipeline assignment
- Need a strategy to find a fast pipeline implementation at runtime

Our Codesign Environment

- Pipeline Selection: choosing and ordering components
- Pipeline Assignment: assigning pipelines to minimize overall latency with the efficient use of software and FPGA
- Pipeline Compilation: creating image processing pipelines dynamically
- Pipeline Execution: executing image processing pipelines



Goal: If pipeline selection is left to the image analyst, can the other three steps be performed automatically at runtime?

Four Shortcomings in Codesign

- Applications are configured statically
  - Design is not sensitive to user changes
- FPGA-based tools do not account for overhead costs
  - Latency is underestimated
- Partition bound too early
  - Interface between HW and SW is hard coded
- Interface changes too costly
  - System code needs extensive rewrites

Four Challenges to Codesign

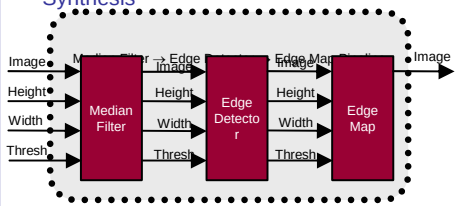
- Combining two design processes
  - Unify implementation languages
- Partitioning design
  - The pipeline assignment problem
- Interfacing hardware and software
  - Abstract communication layer and runtime interface synthesis
- Choosing a target architecture
  - One FPGA and one GPP

SW/HW Runtime Procedural Partitioning Tool

- Solves PA within either fixed or adaptive time limit based on user's choice
- Chooses an algorithm to solve PA based on pipeline size

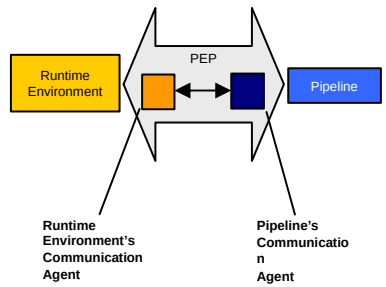
Optimization Method	Fixed	Adaptive
Dynamic Programming	1-7	1-15
1-Opt Tabu Search with Greedy	8,9	--
1-Opt Tabu Search with All Hardware	10-20	16-20

Runtime Interfacing for Pipeline Synthesis



- Builds executable pipeline from PA solution
- Connects the appropriate implementations so that the coupling costs are satisfied

Packet Exchange Platform



- Separates pipeline from runtime environment
- Makes communication abstract and generic

*Absolute Relative Error (ARE) = |(measured latency - predicted latency) / measured latency|

A Two Component Pipeline

- Median Filter→ Histogram
- Image size of 40185 pixels

PA	Predicted Latency w/ overhead (ms)	Predicted Latency w/o overhead (ms)	Actual Latency (ms)	ARE* w/ overhead	ARE* w/o overhead
sw/sw	2509	2509	2141	0.1719	0.1719
sw/hw	4905	2516	3967	0.2365	0.3658
hw_1/sw	2864	376	2975	0.0373	0.8736
hw_2/sw	2852	392	3141	0.0920	0.8752
hw_3/sw	3036	577	3004	0.0107	0.8079
hw_1/hw	2896	399	3042	0.0480	0.8688
hw_2/hw	2884	584	2803	0.0289	0.7917

Random Pipeline Test

- Forty test pipelines of different lengths were run in the Dynamo system for the best latency solution
- Image size of 40185 pixels
- Average ARE: 23% with overhead, 70% without

Test	Predicted Latency w/ overhead (ms)	Predicted Latency w/o overhead (ms)	Actual Latency (ms)	ARE* w/ overhead	ARE* w/o overhead
1	1111	1111	1309	0.1513	0.1513
5	2902	375	3169	0.0843	0.8817
10	3362	743	3571	0.0585	0.7919
15	4509	1411	4789	0.0585	0.7054
20	4849	1701	5955	0.1857	0.7144
25	4928	1785	6012	0.1803	0.7031
30	5297	2114	8560	0.3812	0.7530
35	6006	2575	10922	0.4501	0.7642
40	7289	3450	12217	0.4034	0.7176

Future Work

- Extend the pipeline assignment problem for FPGA devices:
  - in a network of workstations
  - with embedded processors
- Extend the pipeline assignment problem's objectives to include power minimization
- Extend the latency model to include an estimation of the error for better accuracy

Publications

Dynamo: A Runtime Partitioning System, L. A. Smith King, Miriam Leeser, and Heather Quinn, The 2004 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'04), pp. 145-151, Las Vegas, Nevada, June 21-24, 2004.

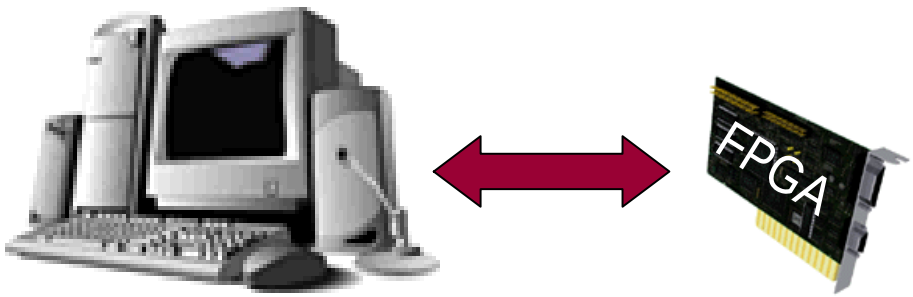
Runtime Assignment of Reconfigurable Hardware Components for Image Processing Pipelines, Heather Quinn, L.A. Smith King, Miriam Leeser, and Waleed Meleis, 2003 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 173-182, Napa, CA, April 8-11, 2003.

Dynamo: A Runtime Codesign Environment

Dr. Heather Quinn, Dr. Miriam Leeser
Northeastern University
hquinn, mel@ece.neu.edu

Dr. Laurie Smith King
College of the Holy Cross
la@mathcs.holycross.edu

On this Environment:



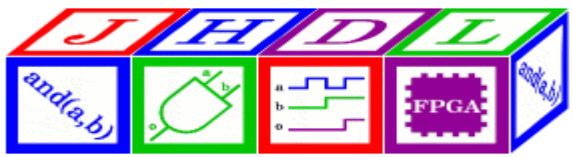
Which implementations to use?

How to minimize overall latency?

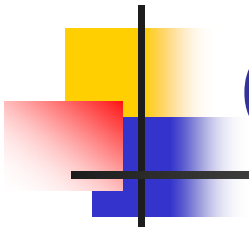
When to use FPGA?

How to change the pipeline or interfaces dynamically?

Run this pipeline:



Median Filter
&
Edge Detection



Goals

Predict pipeline latency accurately:

- Model includes all overhead costs of using the FPGA

Implement low latency solutions:

- Problem space quickly searched

Build and execute pipelines at runtime:

- User is not impacted

Dynamo System Overview

Input: The pipeline specification, an image, and a device name

Output: Results from executing pipeline on the image

Libraries: ipBLOC

Model: Pipeline Assignment